Enterprise Client Technical Overview

This paper talks about the following aspects of CaptainCasa Enterprise Client:

- Architectural Concepts and
- Development Process

Readers should have some general knowledge about Rich Client architectures.

Architectural Concepts

The Client itself

The client part of CaptainCasa Enterprise Client is a generic, browser-like component that is run on a user's frontend machine.

- **"Generic"** means that it does not know about application semantics – it only knows about rendering graphical components and transferring data and events in and out.
- **"Browser-like"** means that it receives a layout in form of an XML page (browser: HTML page), it renders the page using Java Swing graphical components and sends back data changes and events to the server. Like a normal browser it uses the http protocol to send request to the server and to receive back XML-responses.

The rendering of XML pages is optimized for "delta rendering". This means: the browser always checks, when receiving a new layout from the server, what to take over from the currently rendered page. Example: if the page stays the same, but only data inside the page is changed, then no rendering at all is done – just data values are updated inside a page.

This delta rendering is much less extensive from the performance point of view than e.g. a complete re-rendering approach, as is done in normal browsers.

What are the advantages behind this concept?

- The browser is fast and optimized for supporting scenarios in which a user processes one page many times.
- The browser is one generic browser installation and does not know about the application semantics behind a page.
- The XML page definition is open to be produced by any server-side web framework: from PHP to .Net to J2EE. Currently a default implementation is available based on JSF (Java Server Faces, a part of J2EE).
- Firewall and other settings – which are quite often a problem when working with Java frontend software – are no problem at all. Firewalls just need to be configured in the same way as they are configured to support web browsers.
- The way a server-side application talks to the frontend client is via sending an XML layout and receiving back corresponding event requests. There is no client-side coding required – all application logic itself resides on the server side.

The Server Side

As mentioned, any server-side framework that is used for HTML browser processing can also be used to support Enterprise Client processing.

Java Server Faces (JSF) is the dominating standard within the Java environment for server-side UI processing. Though in most cases being used for processing HTML pages, JSF is designed and intended to also be used for other frontend clients.

JSF is built using control libraries, in which the client-side components are abstracted on server-side. A layout definition is represented by a component tree, each node representing a client-side component.

Layouts can be defined in the following ways:

- A JSP (Java Server Page) file is containing an XML definition of the layout that is to be rendered in the client. This JSP page is transformed into a corresponding component tree by JSF.
- A component tree can be directly manipulated by a server-side program. For example a server-side program can dynamically add new components or remove existing components. As consequence, a page can be generated completely dynamically.
- Of course mixed modes are supported as well: starting with a JSP page, the content of the page can be manipulated during the life cycle of the page.

What are the advantages behind using JSF?

- The server-side programming, which connects the client user interface

processing to the business logic, is based on a standard framework. This standard framework is used for plain HTML pages, AJAX-HTML pages, WAP pages (mobile devices) and, as explained, for Enterprise Client pages. Though supporting multiple, different output channels, the server-side processing is one and the same.

- JSF supports all the things required for enterprise application frontends, starting with "high-availability" and "load management" and ending with "internationalization".
- JSF is widely used and accepted in the market of mission-critical applications, there is plenty of knowledge available on the market.

Development Concepts

All development work for CaptainCasa Enterprise Client is done on server side. The server defines the layouts – either statically via JSP definitions or dynamically. And the server holds the programs that manage the data and the events of a page.

The best way to get to know the concepts is by example. After a short overview you will see that "everything is straight and simple".

Overview

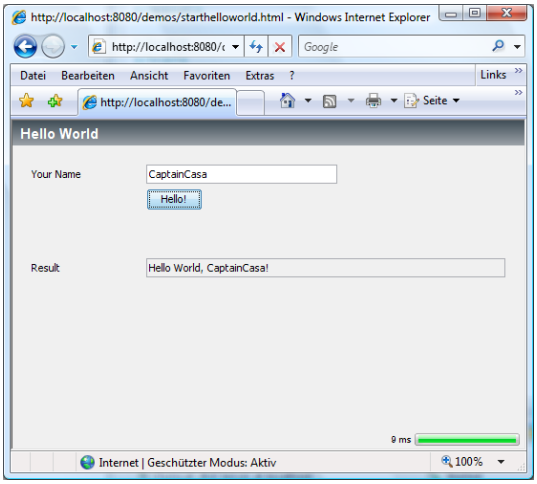
The starting point of a page is the layout definition – a JSP file containing the XML that describes the page that is to be rendered on client side.

The page contains the graphical components and their attributes. Attributes (e.g. the text of a field) are either defined statically within the JSP definition or they are defined dynamically by binding to so-called "managed bean" object.

The "managed bean" class is the logical counterpart of the page. Data which is displayed and edited within a page binds back to corresponding properties of the bean. And events (e.g. "button pressed") bind back to so-called action listener methods.

Example "Hello World"

The following screenshot shows the result:



The user can input a name and will sees some "Hello world"... output after pressing the button.

First have a look onto the JSP layout definition:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@taglib prefix="f"
          uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h"
          uri="http://java.sun.com/jsf/html"%>
<%@taglib prefix="t" uri="/WEB-INF/jtc"%>

<f:view>
<h:form>
<f:subview id="demos_helloworlde_134">

<t:rowtitlebar id="g_2" text="Hello World" />
<t:rowbodypane id="g_3" >
  <t:row id="g_4" >
    <t:label id="g_5"
          text="Your Name"
          width="120" />
    <t:field id="g_6"
          text="#{demoHelloWorld.name}"
          width="200" />
  </t:row>
  <t:rowdistance id="g_7" height="5" />
  <t:row id="g_8" >
    <t:coldistance id="g_9" width="120" />
    <t:button id="g_10"
          actionListener=
                "#{demoHelloWorld.onHello}"
          text="Hello!" />
  </t:row>
  <t:rowdistance id="g_11" height="50" />
  <t:row id="g_12" >
    <t:label id="g_13"
          text="Result"
          width="120" />
    <t:field id="g_14"
          background="#F0F0F0"
          enabled="false"
          text="#{demoHelloWorld.output}"
          width="100%" />
  </t:row>
</t:rowbodypane>

</f:subview>
</h:form>
</f:view>
```

The component part of the page shows that each graphical component that you see in the screenshot is reflected by a corresponding XML tag definition.

The layout definition structures the page into rows, in the rows you find the components. The

components hold attributes (e.g. width). Some of the attributes are statically defined (width=”120”), some of them bind to a managed bean object property (text= ”#{demoHelloWorld.output}”). The button definition binds to a method “onHello”.

The component part is surrounded by some JSP default tags, e.g. the declaration of tag libraries at the beginning, and some tags like “view”,”form” and “subview”.

Now take a look at the class that represents the logic behind the page:

```
package workplace;

import javax.faces.event.ActionEvent;

public class DemoHelloWorld
        implements Serializable
{
    String m_name;
    String m_output;

    public void onHello(ActionEvent ae)
    {
        if (m_name == null)
            m_output = "No name set.";
        else
            m_output = "Hello World, "
                       +m_name+"!";
    }

    public void setName(String value)
    { m_name = value; }
    public String getName()
    { return m_name; }
    public void setOutput(String value)
    { m_output = value; }
    public String getOutput()
    { return m_output; }
}
```

The class provides exactly these properties and method that are bound within the page layout.

Properties are made available by corresponding set/get methods, methods are made available by direct implementation.

Sequence of Operations

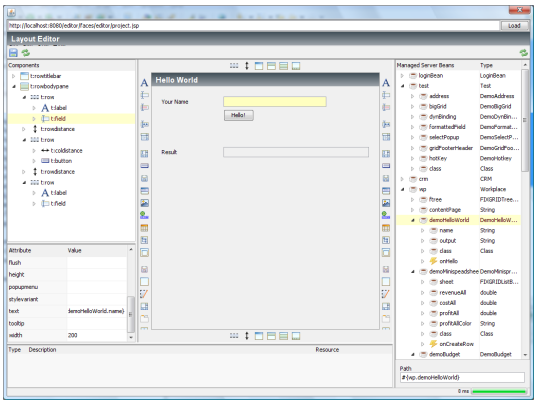
What happens internally in order to bring up the page at a user's frontend?

- The client is loaded onto the user's desktop: this is done the first time accessing the client. Loading can be done using applet or webstart technology – or by explicit installation.
- The client requests the page by sending a corresponding URL to the server.
- The server picks the JSP page from the file system. The server generates an instance of the DemoHelloWorld bean, dependent on the declaration the instance either has session or request scope.
- The server renders the JSP page into XML format and sends this as response back to the client.
- The client receives an XML layout definition and renders its content accordingly.
- The user does some input and then presses the button “Hello!”.
- The client sends a request to the server, the request's data contains both the data changes (user's field input) and the event (button pressed).
- The server receives the request, passes the data changes onto the managed bean object and calls the corresponding “onHello” method.
- The server renders the page again in XML, this time the contained field values have changed due to the “onHello” method execution.
- The XML response is sent back to the client.

This cycle of operations is now repeated on and on again.

Tool Support

CaptainCasa Enterprise Client Designer is a set of tools supporting the creation of layout definitions.



The main tool is the Layout Editor. Here you can define your JSP page definitions by using a WYSIWYG editing environment.

“On the left” you maintain the page. “On the right” you see the managed bean declarations – in order to bind corresponding properties to component attributes.

By the way: all tools that are provided for CaptainCasa Enterprise Client are implemented by using CaptainCasa Enterprise Client as well!

(Some people call this: “Eat you own dog food!”...)

More Complex Scenarios

Of course, “Hello World!” is only sufficient to tell about the basics.

But: the principal architecture does not change when coming to more complex scenarios.

In more complex scenarios you may find:

- More components – covering lists and trees with various contained elements, maybe with right mouse button popup menus and with function key support
- More events – covering drag & drop operations, single and double-clicks, change events from input components, and much more
- Navigation – getting from one page to the next, opening modal or modeless popups
- Aggregation of pages – maybe one page is nested into another page
- Multi language definitions (UTF-8 is the base for all character definitions)

You can find all information how to do this in the Enterprise Client Developers' Guide. And you will find a lot of examples as part of the product delivery.

CaptainCasa GmbH
Hindemithweg 13
D – 69245 Bammental
+49 6223 484147

www.CaptainCasa.com
info@CaptainCasa.com